Date __/__/____

Saathi

Trie

Trie data structure is defined as a Tree based data structure that is used for storing some collection of strings and performing efficient search operations on them. The word Trie is derived from "re Trieval." which means finding something or obtaining it.

Trie data structure follows some property that if two strings have common prefix then they will have the same ancestor in the trie. A trie data structure can be used to sort a collection of strings alphabetically as well as search whether a string with a given prefix is present in the trie or not.

Need of Trie → A Trie data structure is used for storing and retrival of data and the same operations could be done using another data structure which Hash Table but Trie can perform these operations more efficiently than a Hash Table Moreover, Trie has its own advantages over the Hash table. A Trie data structure can be used for prefix-based searching whereas a Hash table can't be used in the same way.

Advantages of Trie Data structure

1. We can efficiently do prefix search with Trie.
2. We can easily print all words in alphabetical orders which is not easily possible with hashing.
3. There is no overhead of Hash functions in a Trie.
4. Searching for a string even in the large collection of

Page No.

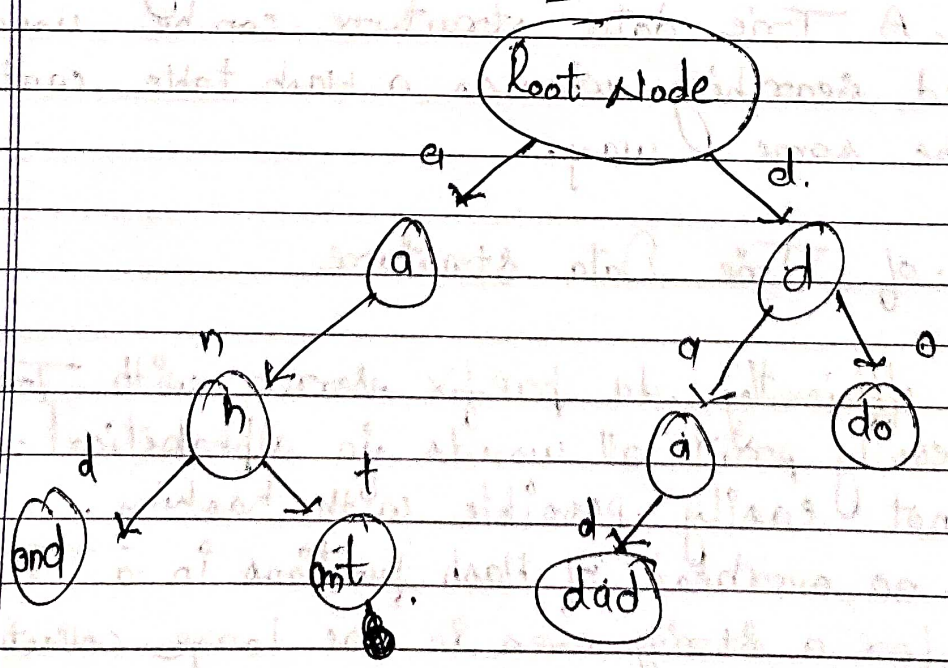
Date ___/___/_____

Strings in a Trie can be done in O(L) T.C. where L is the number of words in the query string.

* Properties of a Trie

1. Each Trie has an empty root node, with links to other nodes.
2. Each node of a Trie represents a string and each edge represents a character.
3. Every node consists of hashmaps or an array of pointers, with each index representing a character and a flag to indicate if any string ends at the current node.
4. Trie data str. can contain any number of characters including alphabets, numbers and special characters.
5. Each path from the root to any node represents a word or string.

Trie



1. and
2. ant
3. dad
4. do

Saathi

Date ___ / ___ / _____

Space / Time Complexity in Trie

Operations	Time Complexity	Auxiliary Space
Insertion	O(n)	O(n*m)
Searching	O(n)	O(1)
Deletion	O(n)	O(1)